1. **INTRODUCTION**

**1.1 Introduction**

The **Organ Donation Management System (ODMS)** is developed mainly for general hospitals (GH), clinics and other health centers to manage the donor registration and user maintenance.

The public can retrieve information about organ donation through this desktop application. People who are interested in organ donation can register themselves through this system.

The application will be processed by the administrator and each donor will receive feedback about their application status. The donor can visit specified medical centers to perform a medical check-up or can ask for a medical assistance at home.

Only administrator has the authority and privileges to print organ list report and total donation report according to district from this system. An analysis study has been done based on the current manual system and all the problems statements and requirements have been identified.

In addition, the improvement part of this system is to help the administrators to easily retrieve the donor’s details. Other than that, it also supports the data integrity for each and every change which is done on to the system. This system also assures the data integrity and helps the management handle the donor’s registration more efficiently.

This Online Organ Donation Management System will help to improve the current situation and overcome the problems that arise nowadays.

**1.2 Problem Formulation**

The current Organ Donation System still using the manual file system which is also known as a simple database. Ledgers and logbooks are wisely used to record the information and evens of the donors. Moreover, the current online system is only for retrieve information about organ donation and donor’s registration. There are fewer functions for administrators where they need to calculate and organized the total registration of donors manually. Rather than that, there is no response or feedback to donors regarding to their registration. This Organ Donation Management System (ODMS) will help the donors, administrators and staff of the organ donation department for a better performance.

1. **DESCRIPTION**
   1. **Uses:**

* To efficiently manage organ donor and recipients profiles.
* Reduce inconsistency in recorded data.
* Help Clinics or hospitals to keep up to date record.
* Archive data for analysis or report generation for human resource management.

**2.2 Application:**

The ODMS can be implemented in various sectors such as:

* Hospital management system where the record section can keep organ transplant data in secure state.
* Government Agencies to keep vigil eyesight over organ transplantations thus ensuring arrest on illegal tissue trafficking.
* Government schemes to promote organ donation.

1. **MODULES**

**3.1 Home:**

In this module, short information about Organ Donation along with the need to be an organ donor is given. It also motivates the donors to voluntarily donate their organs as it is the ultimate humanitarian act of charity.

**3.2 Registration:**

In this module the registration of both donor and receiver will we carried out. Each user need to fill all the required details correctly will get their unique Id type which they will use to log-in into the system.

**3.3 Login:**

In this module the receiver can log into his/her profile. The log-in Id will be one provide during the profile registration. In this module admin login is performed where he can search and delete records.

The login module is further divided into three sub modules.

The modules to pop up will be defined by the log-in Id type.

2A: Receiver login:

This module will allow updates to the receiver profile.

The hospitals, admin will have partial control over such profiles.

2B: Admin login:

These profiles will be internally generated in the system.

Such profiles will not have any registration platform.

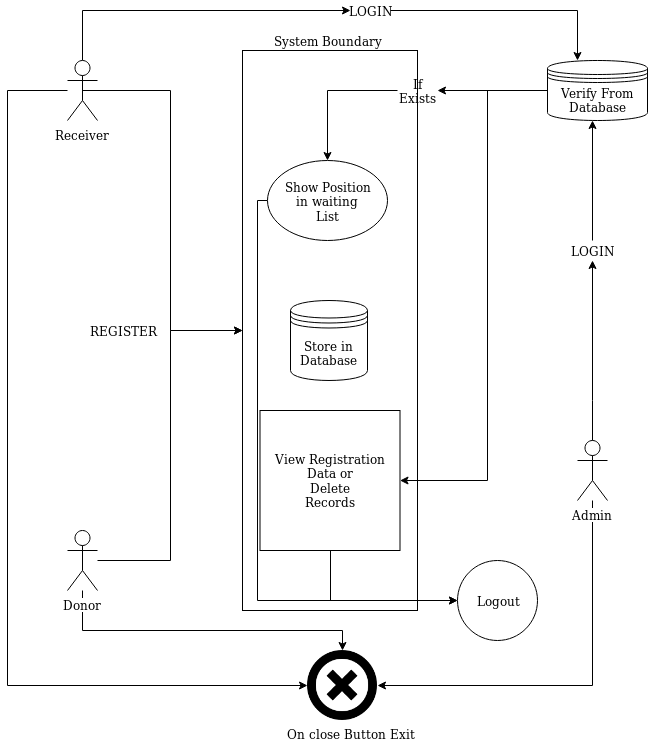
Admin could be hospitals, GC, governing bodies.

**3.4 About**

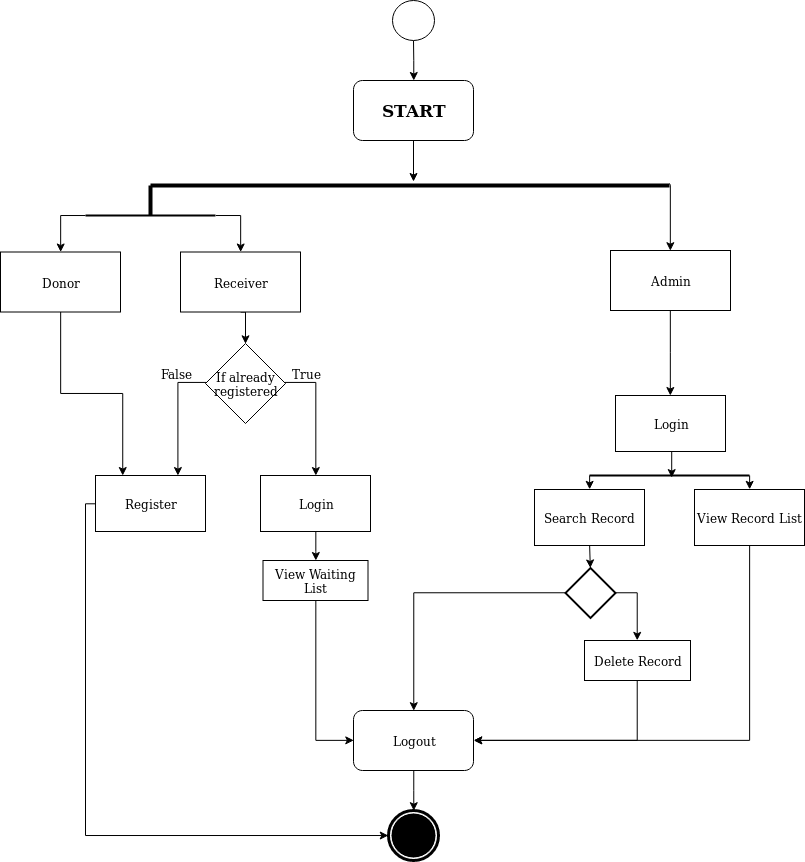
In this module, the name of the project along with the version and copyrights are given.

1. **METHODOLOGY**

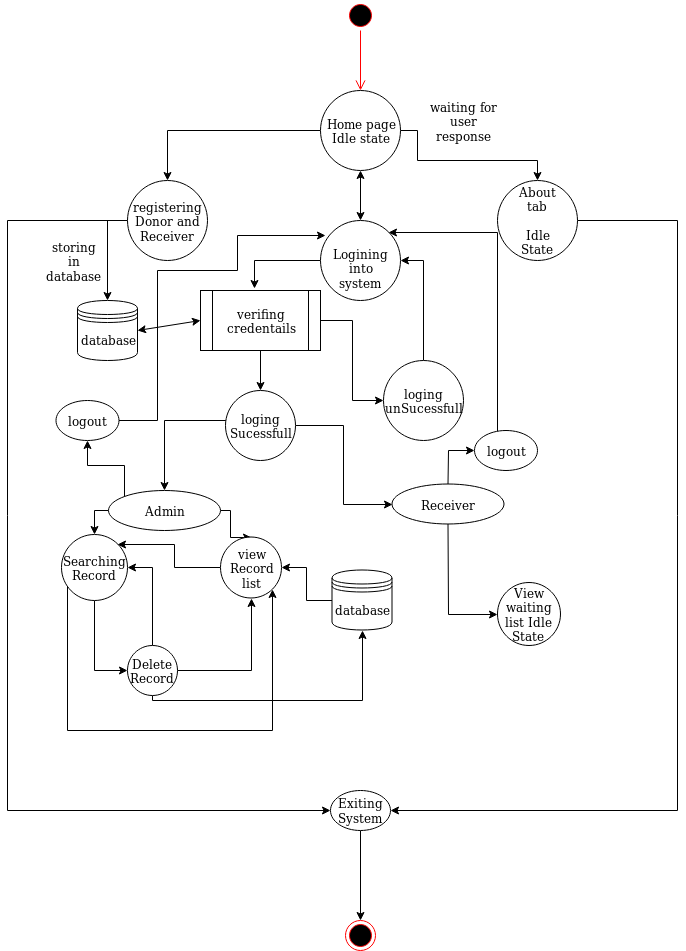
**4.1 Use-Case Diagram**

****

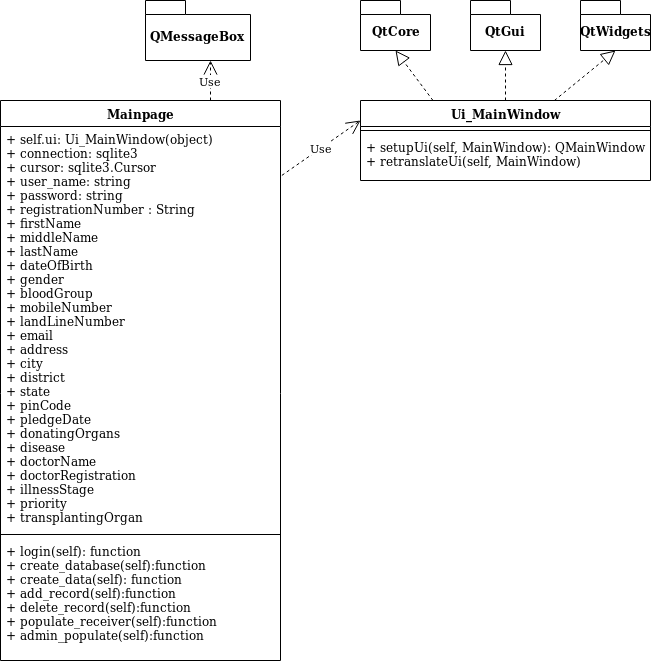
**4.2 Activity Diagram**

****

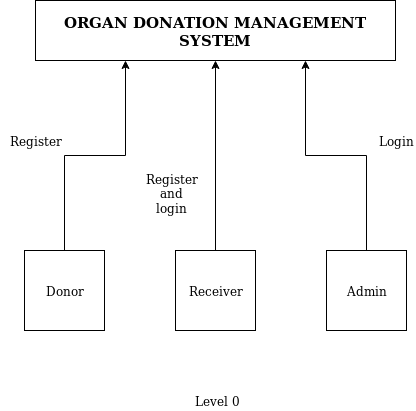
**4.3 State Diagram**

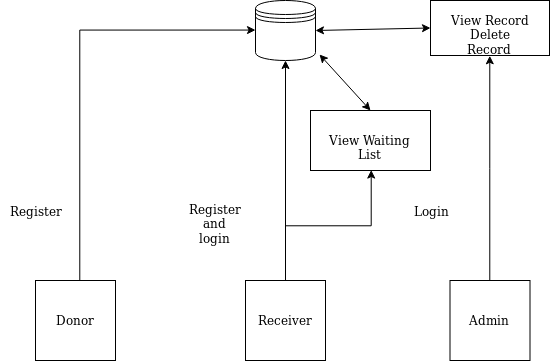
****

**4.4 Class Diagram**

****

* 1. **Data Flow Diagram**

****

****

1. **CODING**

import datetime

import re

import sqlite3

import sys

from PyQt5.QtWidgets import QMessageBox

from PyQt5 import QtWidgets, uic

from PyQt5 import QtGui

class Test:

cursor: sqlite3.Cursor

def \_\_init\_\_(self):

self.ui = uic.loadUi('organ.ui')

self.ui.show()

self.create\_database()

self.ui.loginButton.clicked.connect(self.login)

self.ui.homeTabRegisterButton.clicked.connect(self.tab\_change)

self.ui.formSelecter.currentIndexChanged.connect(self.change\_form)

self.ui.registerTabRegisterButton.clicked.connect(self.add\_record)

self.ui.goToLoginPageReceiver.clicked.connect(self.set\_login\_page)

self.ui.goToLoginPageAdmin.clicked.connect(self.set\_login\_page)

self.ui.deleteRecord.clicked.connect(self.delete\_record)

def login(self):

user\_name: str

password: str

self.ui.statusNameLabel.setText('')

self.ui.statusPasswordLabel.setText('')

try:

user\_name = self.ui.userName.text()

password = self.ui.userPassword.text()

if user\_name == "" and password == "":

raise Exception('both')

if user\_name == "":

raise Exception('user\_name')

if password == "":

raise Exception('password')

except Exception as e:

if str(e) == 'both':

self.ui.statusNameLabel.setText('\* UserName Required')

self.ui.statusPasswordLabel.setText('\* Password Required')

if str(e) == 'user\_name':

self.ui.statusNameLabel.setText('\* UserName Required')

if str(e) == 'password':

self.ui.statusPasswordLabel.setText('\* Password Required')

else:

check = None

if 'Admin' in user\_name:

self.cursor.execute('select \* from loginId where username = ? and password = ?;',

(user\_name, password))

check = self.cursor.fetchall()

if check:

self.populate\_record()

self.ui.changeLoginTypeStack.setCurrentIndex(1)

def create\_database(self):

try:

self.connection = sqlite3.connect('organRecord.db')

self.cursor = self.connection.cursor()

self.create\_data()

self.ui.statusbar.showMessage("Connected to Database")

except Exception as e:

print(e)

self.ui.statusbar.showMessage(" Failed to Connect to Database: Exiting Application")

app.exit()

def create\_data(self):

try:

query = """CREATE TABLE IF NOT EXISTS receiver(registrationNumber VARCHAR(5), firstName VARCHAR(20),middleName VARCHAR(20), lastName VARCHAR(20), dateOfBirth DATE, gender VARCHAR(10), bloodGroup VARCHAR(3), mobileNumber VARCHAR(15), landLineNumber VARCHAR(15), email VARCHAR(100),address VARCHAR(255), city VARCHAR(30), district VARCHAR(30),state varchar(30), pinCode VARCHAR(6),disease VARCHAR(50),doctorName VARCHAR(50), doctorRegistration VARCHAR(30), illnessStage VARCHAR(15),priority VARCHAR(20),transplantingOrgan VARCHAR(10));"""

self.cursor.execute(query)

self.connection.commit()

query = """CREATE TABLE IF NOT EXISTS loginId(userName varchar(10),password varchar(10));"""

self.cursor.execute(query)

query = """SELECT \* FROM loginId;"""

self.cursor.execute(query)

if not self.cursor.fetchone():

query = """INSERT INTO loginId(userName, password) values('Admin','Admin@0');"""

self.cursor.execute(query)

self.connection.commit()

except Exception as e:

print(e)

def add\_record(self):

self.ui.emailVerify.setText(" ")

if self.ui.formSelecter.currentIndex() == 1:

data = []

registration\_number = 'REC1'

self.cursor.execute("select registrationNumber from receiver;")

if self.cursor.fetchone():

self.cursor.execute("select registrationNumber from receiver ORDER BY registrationNumber DESC LIMIT 1;")

old\_registration\_number = self.cursor.fetchone()

registration\_number = int(re.search(r'\d+', old\_registration\_number[0]).group())

registration\_number = 'REC' + str(registration\_number + 1)

data.append(registration\_number)

data.append(self.ui.fristName.text())

data.append(self.ui.middleName.text())

data.append(self.ui.lastName.text())

date\_of\_birth = self.ui.dateOfBirth.date().toPyDate()

data.append(date\_of\_birth.strftime("%d-%m-%Y"))

if self.ui.radioFemale.isChecked():

data.append('Female')

if self.ui.radioMale.isChecked():

data.append('Male')

else:

data.append('Other')

data.append(self.ui.bloodGroup.currentText())

data.append(self.ui.mobileNumber.text())

data.append(self.ui.landLineNumber.text())

data.append(self.ui.email.text())

s = str(self.ui.email.text())

if not s.endswith('@gmail.com'):

self.ui.emailVerify.setText("Invalid Email id")

return False

data.append(self.ui.permanantAddress.toPlainText())

data.append(self.ui.addCity.text())

data.append(self.ui.addState.currentText())

data.append(self.ui.addDistrict.text())

data.append(self.ui.addPincode.text())

data.append(self.ui.patientDisease.text())

data.append(self.ui.doctorName.text())

data.append(self.ui.doctorRegistration.text())

data.append(self.ui.illnessStage.currentText())

data.append(self.ui.patientPriority.currentText())

data.append(self.ui.transplantOrgan.currentText())

query = """INSERT INTO receiver(registrationNumber , firstName, middleName , lastName , dateOfBirth , gender , bloodGroup , mobileNumber , landLineNumber , email , address , city , district , state, pinCode, disease, doctorName, doctorRegistration,illnessStage,priority, transplantingOrgan) VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?);"""

self.cursor.execute(query, data)

self.connection.commit()

word = data[1]

word2 = data[2]

word = word[2:5] + word2[2:5]

response = add\_sucess(registration\_number, word)

self.add\_login\_id(registration\_number, word)

if response:

self.renew\_application()

def add\_login\_id(self, registration, word):

query = """insert into loginId (userName, password) values(?,?);"""

self.cursor.execute(query, (registration, word))

return True

def populate\_receiver(self, user\_id):

try:

query = """ select registrationNumber, transplantingOrgan, priority, bloodGroup, doctorName from receiver;"""

self.cursor.execute(query)

waiting\_table = self.cursor.fetchall()

required\_rows = len(waiting\_table)

self.ui.receiverLoginStackTable.setRowCount(required\_rows)

x = -1

for i in waiting\_table:

x = x + 1

for j in range(0, 5):

self.ui.receiverLoginStackTable.setItem(x, j, QtWidgets.QTableWidgetItem(i[j]))

query = """select transplantingOrgan from receiver where registrationNumber =?;"""

self.cursor.execute(query, (user\_id,))

value = self.cursor.fetchall()

query = """select registrationNumber from receiver where transplantingOrgan = ? ; """

self.cursor.execute(query, value[0])

organ = self.cursor.fetchall()

count = 1

for i in organ:

if i[0] is None:

continue

elif i[0] == user\_id:

self.ui.place.setText(str(count))

else:

count = count + 1

except Exception as e:

print(e)

if \_\_name\_\_ == '\_\_main\_\_':

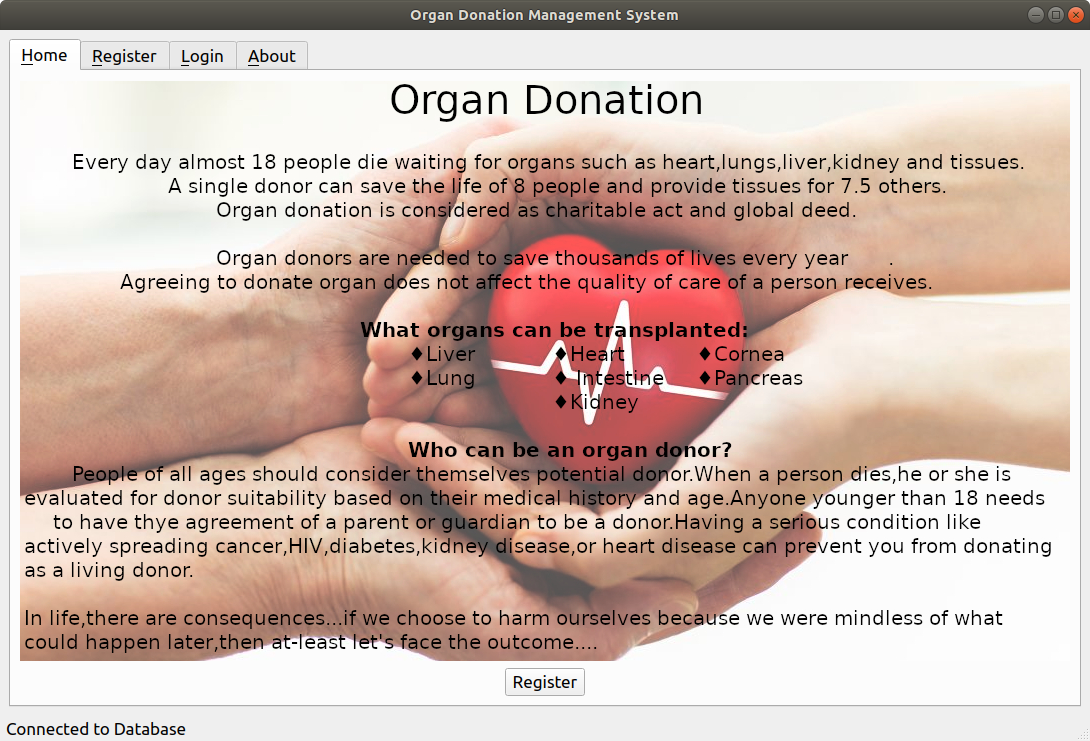
app = QtWidgets.QApplication(sys.argv)

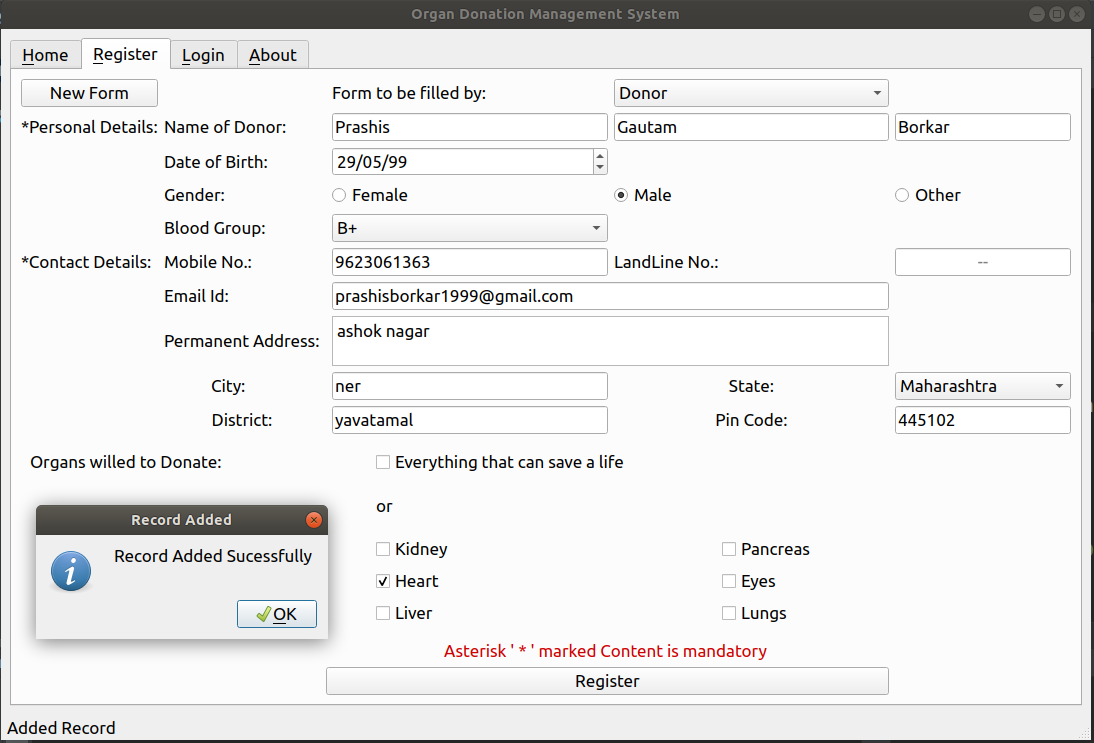
win = Test()

sys.exit(app.exec\_())

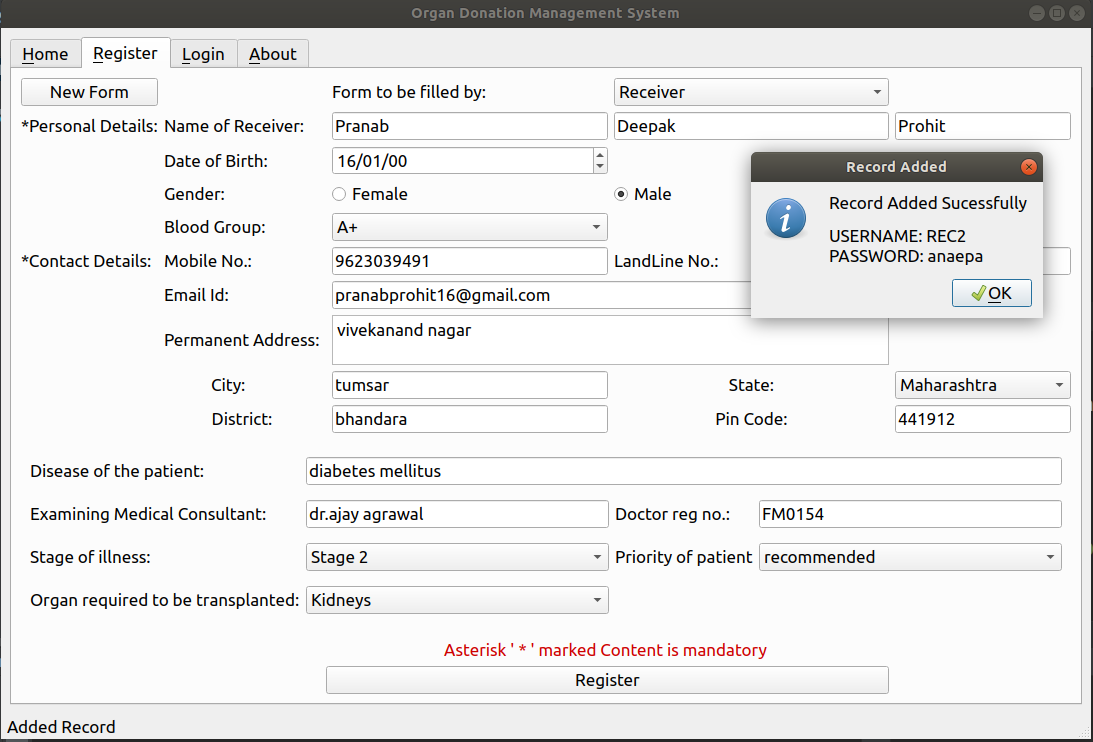
**6. SCREENSHOTS**

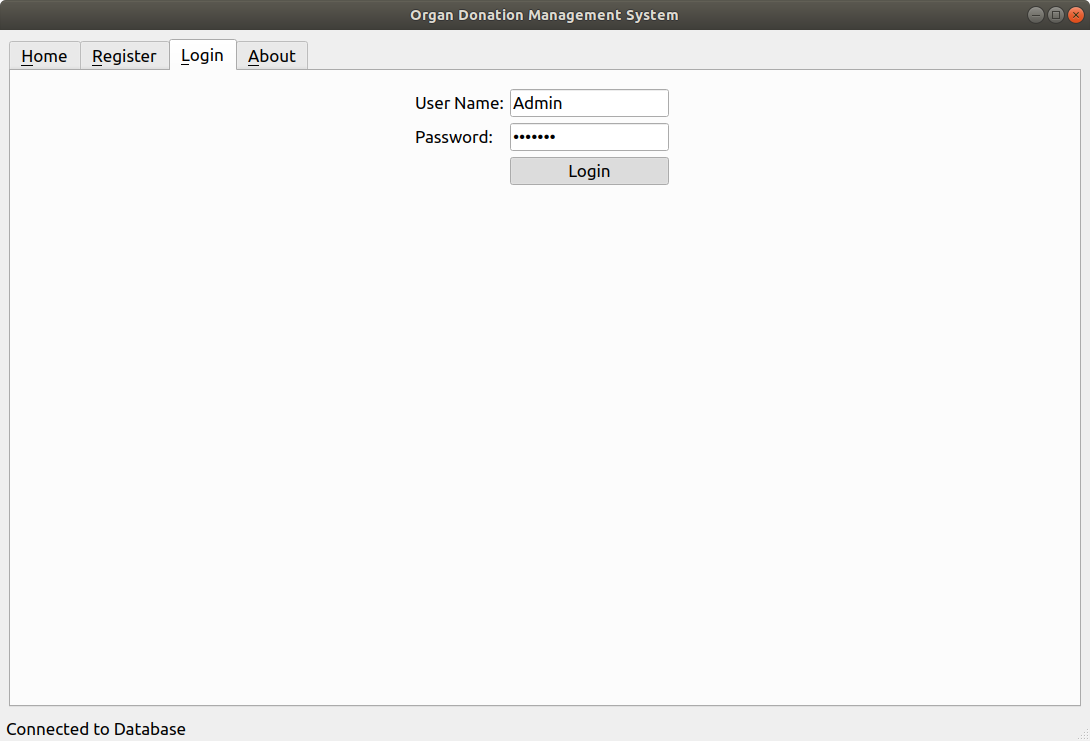
**6.1 Home Page**

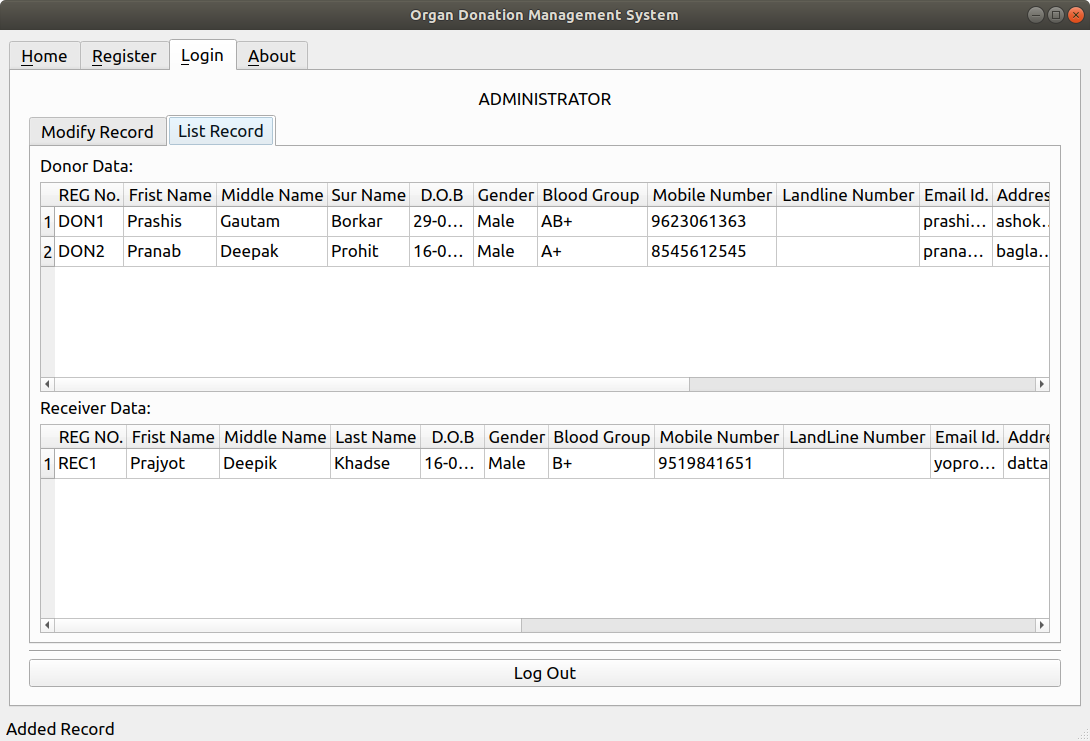


**6.2 Donor Registration**

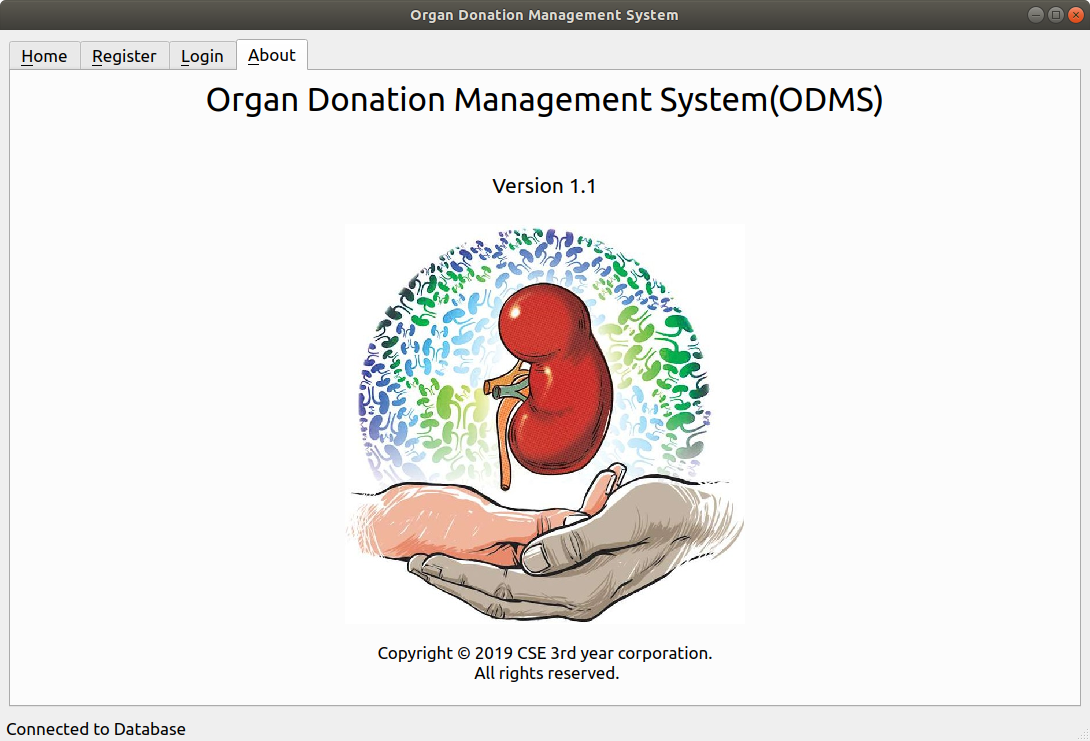
**6.3 Receiver Registration**



**6.4 Admin Login**



**6.5 About Window**



**7. SYSTEM REQUIREMENTS**

**7.1 Software Requirements**

The minimum software requirement specifications for developing this project are as follows:

Programming Language: Python 3.6

Development Environment: PyCharm Community Edition IDE of Jet Brains

Backend Database: SQLite 3

UI Interfacing: Qt Designer

VCS (Version Control System): GITHUB of Microsoft Corporation.

**7.2 Hardware Requirements**

The minimum hardware requirement specifications for developing this project are as follows:

Processor: Intel core Duo 1.2 GHz or AMD Ryzen 3 2.0 GHz or greater

Hardisk: 20 MB of free space

Ram: 2 GB DDR3 or greater

Host OS: Windows 7 SP1 or Ubuntu Linux 16.07 or greater

**8. ADVANTAGES AND DISADVANTAGES**

**8.1 Advantages**

* Efficient management of Organ Donor and Receiver Profiles
* Decrease in Data Redundancy
* Increase in data consistency
* Implementation of Government Campaigns become Flawless
* Hospitals and Organ Donation Centre can have centralized Records

**8.2 Disadvantages**

* Users should have some prior knowledge of handling Computers
* Authorized users should only have access to system
* Non maintenance of System can lead to data corruption or data lose.

**9. CONCLUSION**

Hence we have successfully developed Organ Donation Management System using the python 3.7 programming language. This software meets the objective and goals proposed earlier in the report. The user will find it useful compared to any other software in the society as it provides the better features with lesser complexities and better user interfaces.

Also during this project we have learned process of developing the software such as software documentation, team-work.

**10. REFERENCES**

* <https://en.wikipedia.org/wiki/Organ_donation>
* <https://www.mohanfoundation.org/>
* <https://www.python.org/>
* <https://riverbankcomputing.com/software/pyqt/intro>
* <https://www.youtube.com/results?search_query=sqlite3+python+tutorial>
* [https://sci-hub.tw/https://ieeexplore.ieee.org/document/7930122](https://sci-hub.tw/https:/ieeexplore.ieee.org/document/7930122)